create a python script for create a vm on comput engine with ubuntu image and deploy to google cloud using python script

**ChatGPT said:**

To create a VM on Google Cloud Engine with an Ubuntu image and deploy it using a Python script, you'll need to use the google-cloud-sdk along with google-api-python-client and google-auth libraries. Here is a step-by-step guide to create such a script.

**Prerequisites:**

1. **Install Google Cloud SDK**: You need the Google Cloud SDK installed on your local machine or server, which can be done from here: Google Cloud SDK Installation
2. **Enable Google Cloud API**:
   * Enable the "Compute Engine API" on your Google Cloud Console: Enable Compute Engine API
   * Ensure that billing is enabled for your Google Cloud project.
3. **Set up Authentication**:
   * You need a service account with the necessary permissions. You can create a service account key from the Google Cloud Console.
   * Export the path of your service account credentials file as an environment variable like so:

bash

CopyEdit

export GOOGLE\_APPLICATION\_CREDENTIALS="/path/to/your-service-account-file.json"

1. **Install Required Python Libraries**: You can install the necessary libraries using pip:

bash

CopyEdit

pip install google-api-python-client google-auth google-auth-httplib2 google-auth-oauthlib

**Python Script to Create VM with Ubuntu Image:**

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

# Set your project ID and zone here

PROJECT\_ID = 'your-project-id'

ZONE = 'us-central1-a' # Or another zone you prefer

INSTANCE\_NAME = 'ubuntu-vm'

IMAGE\_PROJECT = 'ubuntu-os-cloud'

IMAGE\_FAMILY = 'ubuntu-2004-lts' # Ubuntu 20.04 LTS image

# Authenticate and create the compute client

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [{

'network': 'global/networks/default',

'accessConfigs': [{'type': 'ONE\_TO\_ONE\_NAT', 'name': 'External NAT'}]

}],

'tags': {

'items': ['http-server', 'https-server']

},

}

try:

# Create the VM instance

print(f"Creating VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

**Explanation:**

1. **Setup the Compute Engine Client**:
   * We use google.auth.default() to authenticate using the default credentials, which should already be set up from the GOOGLE\_APPLICATION\_CREDENTIALS environment variable.
   * googleapiclient.discovery.build helps in creating the Compute Engine API client.
2. **VM Configuration**:
   * We specify the instance name, machine type, disk configuration, and the network settings.
   * The Ubuntu image is selected from the ubuntu-os-cloud project and the ubuntu-2004-lts family. You can change this based on the version you want.
3. **Creating the VM**:
   * The instances().insert method is used to send a request to Google Cloud to create the VM.
   * After submitting the request, we wait for the operation to complete using the zoneOperations().get() method.
4. **Waiting for Operation**:
   * The wait\_for\_operation function continuously checks the status of the operation to ensure the VM is created successfully.
5. **Execution**:
   * Run the script, and it will create a VM in your specified project and zone with the Ubuntu image.

**Additional Notes:**

* You can modify the machine type, disk size, image version, and network settings based on your needs.
* If you want to install software or configure the VM further, you can either configure it manually after deployment or use startup scripts to automate this process.

This script provides a simple, straightforward way to deploy an Ubuntu VM on Google Cloud using Python.

4o

**You said:**

manifest-zephyr-457115-v2 - use this project id please

**ChatGPT said:**

Sure! I'll update the Python script with the project ID manifest-zephyr-457115-v2. Here's the updated script:

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import time

# Set your project ID and zone here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Updated project ID

ZONE = 'us-central1-a' # Or another zone you prefer

INSTANCE\_NAME = 'ubuntu-vm'

IMAGE\_PROJECT = 'ubuntu-os-cloud'

IMAGE\_FAMILY = 'ubuntu-2004-lts' # Ubuntu 20.04 LTS image

# Authenticate and create the compute client

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [{

'network': 'global/networks/default',

'accessConfigs': [{'type': 'ONE\_TO\_ONE\_NAT', 'name': 'External NAT'}]

}],

'tags': {

'items': ['http-server', 'https-server']

},

}

try:

# Create the VM instance

print(f"Creating VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

**Key Changes:**

* The PROJECT\_ID is now set to manifest-zephyr-457115-v2.

**Running the Script:**

Ensure you've authenticated with Google Cloud and set your credentials as mentioned earlier. When you run the script, it will create a VM in your manifest-zephyr-457115-v2 project.

Let me know if you need further modifications!

4o

**You said:**

how to run this script

**ChatGPT said:**

To run the Python script, you'll need to follow these steps:

**Step 1: Install Python (if not already installed)**

Make sure Python 3.6+ is installed on your machine. You can download Python from [here](https://www.python.org/downloads/).

You can check if Python is installed by running this command in the terminal/command prompt:

bash

CopyEdit

python --version

Or for some systems:

bash

CopyEdit

python3 --version

**Step 2: Set up Google Cloud Authentication**

Before running the script, ensure you have set up authentication for your Google Cloud account.

1. **Create a Service Account and Download the Key**:
   * Go to the Google Cloud Console.
   * Navigate to **IAM & Admin** > **Service Accounts**.
   * Create a new service account and grant it the **Compute Admin** role.
   * After creating the account, download the **JSON** key file.
2. **Set the GOOGLE\_APPLICATION\_CREDENTIALS environment variable**: This variable allows the script to authenticate using the service account key. Run the following command in your terminal (replace /path/to/your-service-account-file.json with the actual path to the JSON file):

For Linux/Mac:

bash

CopyEdit

export GOOGLE\_APPLICATION\_CREDENTIALS="/path/to/your-service-account-file.json"

For Windows (CMD):

cmd

CopyEdit

set GOOGLE\_APPLICATION\_CREDENTIALS=C:\path\to\your-service-account-file.json

For Windows (PowerShell):

powershell

CopyEdit

$env:GOOGLE\_APPLICATION\_CREDENTIALS="C:\path\to\your-service-account-file.json"

**Step 3: Install Required Python Libraries**

Install the necessary Python libraries. Open your terminal and run:

bash

CopyEdit

pip install google-api-python-client google-auth google-auth-httplib2 google-auth-oauthlib

If you're using Python 3 and the above doesn't work, use:

bash

CopyEdit

pip3 install google-api-python-client google-auth google-auth-httplib2 google-auth-oauthlib

**Step 4: Save the Python Script**

Save the script as a .py file, for example create\_vm.py.

**Step 5: Run the Script**

Once the setup is complete, you can run the script from your terminal or command prompt.

In the terminal, navigate to the directory where the script is saved and run:

bash

CopyEdit

python create\_vm.py

Or, if using Python 3:

bash

CopyEdit

python3 create\_vm.py

The script will start creating the VM in Google Cloud using the provided project ID (manifest-zephyr-457115-v2), zone (us-central1-a), and the Ubuntu 20.04 LTS image.

**Step 6: Check Google Cloud Console**

After the script completes, check the Google Cloud Console for the created VM under the **Compute Engine** section. You should see your newly created VM there.

If the script encounters any issues, it will output error messages that can help you debug the problem.

4o

**You said:**

import os import google.auth from googleapiclient.discovery import build from googleapiclient.errors import HttpError import time # Set your project ID and zone here PROJECT\_ID = 'manifest-zephyr-457115-v2' # Updated project ID ZONE = 'us-central1-a' # Or another zone you prefer INSTANCE\_NAME = 'ubuntu-vm' IMAGE\_PROJECT = 'ubuntu-os-cloud' IMAGE\_FAMILY = 'ubuntu-2004-lts' # Ubuntu 20.04 LTS image # Authenticate and create the compute client def create\_instance(): # Initialize the Compute Engine client credentials, project = google.auth.default() compute = build('compute', 'v1', credentials=credentials) # Define the machine type and image machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}" # Define the configuration for the VM instance config = { 'name': INSTANCE\_NAME, 'machineType': machine\_type, 'disks': [{ 'boot': True, 'autoDelete': True, 'initializeParams': { 'sourceImage': source\_disk\_image } }], 'networkInterfaces': [{ 'network': 'global/networks/default', 'accessConfigs': [{'type': 'ONE\_TO\_ONE\_NAT', 'name': 'External NAT'}] }], 'tags': { 'items': ['http-server', 'https-server'] }, } try: # Create the VM instance print(f"Creating VM instance {INSTANCE\_NAME}...") operation = compute.instances().insert( project=PROJECT\_ID, zone=ZONE, body=config ).execute() # Wait for the operation to complete wait\_for\_operation(compute, operation) print(f"VM instance {INSTANCE\_NAME} created successfully.") except HttpError as err: print(f"An error occurred: {err}") return def wait\_for\_operation(compute, operation): print("Waiting for operation to complete...") while True: result = compute.zoneOperations().get( project=PROJECT\_ID, zone=ZONE, operation=operation['name'] ).execute() if result['status'] == 'DONE': if 'error' in result: raise Exception(f"Error: {result['error']}") else: print("Operation completed successfully.") break else: print("Operation is still in progress...") time.sleep(10) if \_\_name\_\_ == "\_\_main\_\_": create\_instance() - user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python create\_vm.py Traceback (most recent call last): File "D:\agcp\create\_vm.py", line 82, in <module> create\_instance() ~~~~~~~~~~~~~~~^^ File "D:\agcp\create\_vm.py", line 17, in create\_instance credentials, project = google.auth.default() ~~~~~~~~~~~~~~~~~~~^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\auth\\_default.py", line 685, in default raise exceptions.DefaultCredentialsError(\_CLOUD\_SDK\_MISSING\_CREDENTIALS) google.auth.exceptions.DefaultCredentialsError: Your default credentials were not found. To set up Application Default Credentials, see https://cloud.google.com/docs/authentication/external/se t-up-adc for more information.

**ChatGPT said:**

The error message you're seeing indicates that your Google Cloud credentials are not set up or not found by the script. Specifically, it's pointing to a missing **Application Default Credentials (ADC)** configuration.

To resolve this issue, you need to authenticate using your Google Cloud service account credentials and set the environment variable for ADC.

**Here's how you can fix this:**

**Step 1: Create a Service Account and Download the Key**

1. **Go to the Google Cloud Console**:
   * Open Google Cloud Console.
2. **Create a Service Account**:
   * Navigate to **IAM & Admin** > **Service Accounts**.
   * Click **Create Service Account**.
   * Provide a name (e.g., vm-creator) and a description.
   * Under **Role**, choose **Compute Admin** and any other roles that your project might need.
   * Click **Create**.
3. **Create and Download the Key**:
   * Once the service account is created, click the three dots next to it and choose **Create Key**.
   * Select **JSON** and click **Create**. This will download a JSON key file to your machine.
   * Save the file in a secure location.

**Step 2: Set Up Authentication**

Now that you have your service account key file, you need to tell your script to use this file for authentication.

1. **Set the GOOGLE\_APPLICATION\_CREDENTIALS Environment Variable**:

You need to set an environment variable GOOGLE\_APPLICATION\_CREDENTIALS to the path of the downloaded JSON key file.

* + **For Linux/macOS**: Open your terminal and run:

bash

CopyEdit

export GOOGLE\_APPLICATION\_CREDENTIALS="/path/to/your-service-account-file.json"

* + **For Windows** (Command Prompt):

cmd

CopyEdit

set GOOGLE\_APPLICATION\_CREDENTIALS=C:\path\to\your-service-account-file.json

* + **For Windows** (PowerShell):

powershell

CopyEdit

$env:GOOGLE\_APPLICATION\_CREDENTIALS="C:\path\to\your-service-account-file.json"

Replace /path/to/your-service-account-file.json or C:\path\to\your-service-account-file.json with the actual path to your JSON key file.

You can also set the environment variable permanently by adding it to your shell's profile (e.g., .bashrc or .zshrc on Linux/Mac).

**Step 3: Verify Authentication**

To ensure that your authentication is set up correctly, you can run the following command to check if the credentials are working:

bash

CopyEdit

gcloud auth application-default print-access-token

If it prints an access token without any errors, you're good to go.

**Step 4: Run the Python Script**

Now that your authentication is set up, you can run the Python script again. Open your terminal in the directory where your Python script is located and run:

bash

CopyEdit

python create\_vm.py

Or if you're using Python 3:

bash

CopyEdit

python3 create\_vm.py

**Step 5: Verify the VM in Google Cloud Console**

After the script completes, check the **Compute Engine** section in the Google Cloud Console. You should see the newly created VM instance listed.

Let me know if you run into any other issues!

4o

**You said:**

key is in downaload folder

**ChatGPT said:**

Great! Since your service account key is in the **Downloads** folder, you just need to set the GOOGLE\_APPLICATION\_CREDENTIALS environment variable to point to the exact path of your key file.

**Step 1: Set the Environment Variable**

To use the downloaded key file, follow these steps depending on your operating system:

**For Windows (Command Prompt):**

1. Open the **Command Prompt** (CMD) and run the following command, replacing the path with the correct one to your JSON file. For example:

cmd

CopyEdit

set GOOGLE\_APPLICATION\_CREDENTIALS=C:\Users\user121\Downloads\your-service-account-file.json

Be sure to replace your-service-account-file.json with the actual filename of the key you downloaded.

**For Windows (PowerShell):**

1. Open **PowerShell** and run the following command:

powershell

CopyEdit

$env:GOOGLE\_APPLICATION\_CREDENTIALS="C:\Users\user121\Downloads\your-service-account-file.json"

Again, make sure to replace your-service-account-file.json with the actual file name.

**For Linux/Mac:**

1. Open the **Terminal** and run the following command:

bash

CopyEdit

export GOOGLE\_APPLICATION\_CREDENTIALS="/home/username/Downloads/your-service-account-file.json"

Replace /home/username/Downloads/your-service-account-file.json with the correct path to your key file. You can find the path by navigating to the **Downloads** folder and using the pwd command (on Linux/Mac).

**Step 2: Run the Python Script Again**

Once the environment variable is set, go ahead and run the Python script again.

In the terminal, navigate to the folder containing the script and execute it:

bash

CopyEdit

python create\_vm.py

Or if you're using Python 3:

bash

CopyEdit

python3 create\_vm.py

**Step 3: Verify the VM Creation**

Once the script completes, check the **Google Cloud Console** under **Compute Engine** to see your newly created VM.

Let me know if you encounter any other issues!

4o

**You said:**

user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ set GOOGLE\_APPLICATION\_CREDENTIALS=C:\Users\user121\Downloads\deep-gcp.json user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ set GOOGLE\_APPLICATION\_CREDENTIALS=C:\Users\user121\Downloads\deep-gcp.json^C user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python create\_vm.py Traceback (most recent call last): File "D:\agcp\create\_vm.py", line 82, in <module> create\_instance() ~~~~~~~~~~~~~~~^^ File "D:\agcp\create\_vm.py", line 17, in create\_instance credentials, project = google.auth.default() ~~~~~~~~~~~~~~~~~~~^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\site-packages\google\auth\\_default.py", line 685, in default raise exceptions.DefaultCredentialsError(\_CLOUD\_SDK\_MISSING\_CREDENTIALS) google.auth.exceptions.DefaultCredentialsError: Your default credentials were not found. To set up Application Default Credentials, see https://cloud.google.com/docs/authentication/external/se t-up-adc for more information. user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $

**ChatGPT said:**

The error you're encountering suggests that the environment variable for the **Google Application Default Credentials** (ADC) is still not being correctly recognized by the script. This is likely because the environment variable wasn't set properly for your shell session or it's being overwritten.

Here’s what we need to do to fix the issue:

**Step 1: Correctly Set the Environment Variable in Git Bash**

Since you’re using **Git Bash** (as indicated by MINGW64), the syntax for setting environment variables is different compared to Windows Command Prompt or PowerShell.

Use the following command in **Git Bash** to set the GOOGLE\_APPLICATION\_CREDENTIALS environment variable:

bash

CopyEdit

export GOOGLE\_APPLICATION\_CREDENTIALS="/c/Users/user121/Downloads/deep-gcp.json"

Make sure to use the correct path with /c/ instead of C:\ when using Git Bash.

**Step 2: Verify the Environment Variable is Set**

To check if the environment variable is correctly set, you can run:

bash

CopyEdit

echo $GOOGLE\_APPLICATION\_CREDENTIALS

This should output the path to your deep-gcp.json file. If it prints the correct path, you're good to go.

**Step 3: Run the Script Again**

Once the environment variable is set, run the Python script again:

bash

CopyEdit

python create\_vm.py

or if you're using Python 3:

bash

CopyEdit

python3 create\_vm.py

**Step 4: Verify the VM Creation**

After running the script, go to the Google Cloud Console and check under **Compute Engine** for the created VM.

**Additional Tip:**

If you continue to face issues, it's a good idea to check if the file path has any special characters or spaces that might cause issues in Git Bash. If that's the case, you can try moving the key file to a simpler location, such as C:\gcp\deep-gcp.json, and then set the environment variable accordingly.

Let me know how it goes!

4o

**You said:**

working

**ChatGPT said:**

Great to hear it's working now! 🎉

If you have any more questions or need further assistance with Google Cloud or anything else, feel free to ask. Happy coding! 😊

4o

**You said:**

import os import google.auth from googleapiclient.discovery import build from googleapiclient.errors import HttpError import time # Set your project ID and zone here PROJECT\_ID = 'manifest-zephyr-457115-v2' # Updated project ID ZONE = 'us-central1-a' # Or another zone you prefer INSTANCE\_NAME = 'ubuntu-vm' IMAGE\_PROJECT = 'ubuntu-os-cloud' IMAGE\_FAMILY = 'ubuntu-2004-lts' # Ubuntu 20.04 LTS image # Authenticate and create the compute client def create\_instance(): # Initialize the Compute Engine client credentials, project = google.auth.default() compute = build('compute', 'v1', credentials=credentials) # Define the machine type and image machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}" # Define the configuration for the VM instance config = { 'name': INSTANCE\_NAME, 'machineType': machine\_type, 'disks': [{ 'boot': True, 'autoDelete': True, 'initializeParams': { 'sourceImage': source\_disk\_image } }], 'networkInterfaces': [{ 'network': 'global/networks/default', 'accessConfigs': [{'type': 'ONE\_TO\_ONE\_NAT', 'name': 'External NAT'}] }], 'tags': { 'items': ['http-server', 'https-server'] }, } try: # Create the VM instance print(f"Creating VM instance {INSTANCE\_NAME}...") operation = compute.instances().insert( project=PROJECT\_ID, zone=ZONE, body=config ).execute() # Wait for the operation to complete wait\_for\_operation(compute, operation) print(f"VM instance {INSTANCE\_NAME} created successfully.") except HttpError as err: print(f"An error occurred: {err}") return def wait\_for\_operation(compute, operation): print("Waiting for operation to complete...") while True: result = compute.zoneOperations().get( project=PROJECT\_ID, zone=ZONE, operation=operation['name'] ).execute() if result['status'] == 'DONE': if 'error' in result: raise Exception(f"Error: {result['error']}") else: print("Operation completed successfully.") break else: print("Operation is still in progress...") time.sleep(10) if \_\_name\_\_ == "\_\_main\_\_": create\_instance() - explain the whole script line by line please in easy way

**ChatGPT said:**

Sure! Let's break down the script line by line to explain what each part does in a simple and easy-to-understand way.

**1. Imports**

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import time

* **os**: This module is used to interact with the operating system. However, in this script, it's not used directly, so you could potentially remove it.
* **google.auth**: This is used to authenticate with Google Cloud, so the script can make API requests on your behalf.
* **googleapiclient.discovery**: This module helps us interact with Google's APIs. In this case, it will help us interact with the **Compute Engine API** to create and manage virtual machines (VMs).
* **googleapiclient.errors.HttpError**: This is used to handle errors related to Google API requests.
* **time**: This module allows us to pause the program for a certain amount of time (e.g., to wait for the VM creation process to finish).

**2. Variables for Project and VM Configuration**

python

CopyEdit

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Updated project ID

ZONE = 'us-central1-a' # Or another zone you prefer

INSTANCE\_NAME = 'ubuntu-vm'

IMAGE\_PROJECT = 'ubuntu-os-cloud'

IMAGE\_FAMILY = 'ubuntu-2004-lts' # Ubuntu 20.04 LTS image

* **PROJECT\_ID**: This is the unique ID of your Google Cloud project. It tells the script which project to work with.
* **ZONE**: Google Cloud has different "zones" (locations) where VMs can be created. Here, we're using us-central1-a, which is a common zone in the United States.
* **INSTANCE\_NAME**: This is the name that will be assigned to your VM (in this case, "ubuntu-vm").
* **IMAGE\_PROJECT**: This specifies the Google Cloud project that contains the Ubuntu OS image. In this case, it's ubuntu-os-cloud, a standard project for Google’s Ubuntu images.
* **IMAGE\_FAMILY**: This defines the type/version of the Ubuntu OS image that will be used. ubuntu-2004-lts refers to Ubuntu 20.04 LTS (Long-Term Support).

**3. create\_instance() Function**

python

CopyEdit

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

* **create\_instance()**: This function is responsible for creating the VM.
* **google.auth.default()**: This method automatically retrieves your Google Cloud credentials (from your environment variables) so the script can authenticate.
* **build('compute', 'v1', credentials=credentials)**: This creates a connection to the Google Cloud Compute Engine API, which we need to create the VM.

**4. VM Configuration**

python

CopyEdit

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

* **machine\_type**: This defines the type of VM instance you're creating. The "n1-standard-1" type is a basic VM with 1 vCPU and 3.75 GB of memory. You can change this to a more powerful type if needed.
* **source\_disk\_image**: This specifies the source OS image that will be used for the boot disk. Here, it's using the Ubuntu 20.04 LTS image from the ubuntu-os-cloud project.

**5. VM Configuration Details**

python

CopyEdit

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [{

'network': 'global/networks/default',

'accessConfigs': [{'type': 'ONE\_TO\_ONE\_NAT', 'name': 'External NAT'}]

}],

'tags': {

'items': ['http-server', 'https-server']

},

}

* **config**: This is the configuration dictionary that defines the properties of the VM:
  + 'name': The name of the VM, set to the INSTANCE\_NAME variable.
  + 'machineType': The type of machine (VM) you're creating.
  + 'disks': Defines the boot disk for the VM:
    - 'boot': Set to True, meaning this disk will be the boot disk for the VM.
    - 'autoDelete': Set to True, so this disk will be deleted if the VM is deleted.
    - 'initializeParams': Specifies the image to be used for the boot disk (Ubuntu 20.04).
  + 'networkInterfaces': Defines the network interface for the VM, using the default network and adding an external IP (ONE\_TO\_ONE\_NAT) for internet access.
  + 'tags': Specifies tags for the VM (useful for firewall rules, etc.). Here, it's tagged with http-server and https-server to allow HTTP/HTTPS traffic.

**6. Creating the VM and Handling Errors**

python

CopyEdit

try:

# Create the VM instance

print(f"Creating VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

* **compute.instances().insert()**: This sends a request to Google Cloud to create a new VM based on the configuration defined earlier.
* **operation**: The insert method returns an operation object that we need to wait for to complete.
* **wait\_for\_operation()**: This function (defined later) will handle waiting for the operation to finish.
* **Error Handling**: If an error occurs during the creation of the VM, an exception will be raised, and the error message will be printed.

**7. wait\_for\_operation() Function**

python

CopyEdit

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

* **wait\_for\_operation()**: This function repeatedly checks the status of the operation (VM creation process).
  + **compute.zoneOperations().get()**: This checks the status of the operation (using the operation ID).
  + **if result['status'] == 'DONE'**: Once the status is 'DONE', it means the operation has completed. If there is an error, it raises an exception; otherwise, it prints a success message.
  + **time.sleep(10)**: This makes the script wait for 10 seconds before checking again.

**8. Running the Script**

python

CopyEdit

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

* This line ensures that the create\_instance() function is executed when the script is run.

**Summary**

1. **Authentication**: The script uses Google Cloud's authentication to verify that it has permission to create resources.
2. **VM Configuration**: It sets up a configuration for the VM (name, machine type, disk image, etc.).
3. **VM Creation**: The script sends a request to Google Cloud to create the VM.
4. **Wait for Completion**: It waits until the VM is fully created and provides feedback on success or failure.

Let me know if you have any questions or need further clarification! 😊

4o

**You said:**

now create a private vm - it just dont have public ip = plese crate use python script

**ChatGPT said:**

To create a private VM without a public IP, you can modify the network configuration to avoid assigning an external IP address. Here's an updated version of your script that will create a **private VM**, meaning it won't have an external IP address.

**Key Modifications:**

* Remove the accessConfigs part in the networkInterfaces section, which is responsible for assigning a public IP to the VM.

Here's the updated Python script to create a **private VM**:

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import time

# Set your project ID and zone here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Updated project ID

ZONE = 'us-central1-a' # Or another zone you prefer

INSTANCE\_NAME = 'private-ubuntu-vm'

IMAGE\_PROJECT = 'ubuntu-os-cloud'

IMAGE\_FAMILY = 'ubuntu-2004-lts' # Ubuntu 20.04 LTS image

# Authenticate and create the compute client

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [{

'network': 'global/networks/default',

# No accessConfigs means no public IP address

}],

'tags': {

'items': ['http-server', 'https-server']

},

}

try:

# Create the VM instance

print(f"Creating private VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"Private VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

**Explanation of Changes:**

1. **Network Configuration**:
   * The key change to create a **private VM** is removing the accessConfigs from the networkInterfaces section. This ensures that no external (public) IP is assigned to the VM.

python

CopyEdit

'networkInterfaces': [{

'network': 'global/networks/default',

# No accessConfigs means no public IP address

}],

1. **Private VM**:
   * The VM will now only have an internal IP and will be reachable only within the VPC (Virtual Private Cloud) network. If you need to access it, you'll have to use a **jump box** (a VM with a public IP in the same network) or set up VPN access.

**Additional Notes:**

* **Private VM Access**: Since the VM won't have a public IP, you need to access it through an internal network. Typically, this is done through a **bastion host** (a VM with a public IP) or **VPN** if you're working within a corporate network.
* **Firewall Rules**: Make sure your firewall allows the necessary traffic for the private VM, such as SSH or HTTP/HTTPS, depending on your use case.

**Running the Script:**

* Follow the same process as before. Set up your credentials and run the script in your terminal.

bash

CopyEdit

python create\_vm.py

The script will create a private VM in your Google Cloud project, and you will be able to connect to it using internal methods such as a jump box or a VPN.

Let me know if you need further adjustments or help with the setup! 😊

4o

**You said:**

i have my vpc and private subnet - create a private vm using that - make changes in code

**ChatGPT said:**

If you already have a **VPC** and a **private subnet**, you can create a private VM in that specific VPC and subnet. To do that, you'll need to specify the subnet in the networkInterfaces configuration and ensure that the VM is connected to the correct VPC.

**Steps for Modifying the Script:**

1. **Specify the VPC Network**: You need to reference the VPC network you're using.
2. **Specify the Private Subnet**: You need to define the subnet that the VM should be attached to.
3. **Remove Public IP Configuration**: You want to ensure the VM does not get a public IP address, which is done by not including accessConfigs in the networkInterfaces configuration.

**Updated Python Script**

Assuming you have:

* **VPC** name: your-vpc-name
* **Private Subnet** name: your-private-subnet
* **Region**: us-central1
* **Zone**: us-central1-a

Here's how you can modify your script:

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import time

# Set your project ID, VPC, subnet, and zone here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Updated project ID

ZONE = 'us-central1-a' # The zone for the VM

INSTANCE\_NAME = 'private-ubuntu-vm'

IMAGE\_PROJECT = 'ubuntu-os-cloud'

IMAGE\_FAMILY = 'ubuntu-2004-lts' # Ubuntu 20.04 LTS image

VPC\_NAME = 'your-vpc-name' # Replace with your VPC name

SUBNET\_NAME = 'your-private-subnet' # Replace with your private subnet name

# Authenticate and create the compute client

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

# Define the network and subnet information

network\_interface = {

'network': f'global/networks/{VPC\_NAME}', # VPC name

'subnetwork': f'regions/{ZONE.split("-")[0]}/subnetworks/{SUBNET\_NAME}', # Subnet name and region

# No accessConfigs means no public IP address

}

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [network\_interface], # Use the network interface with the private subnet

'tags': {

'items': ['http-server', 'https-server'] # Add tags as needed

},

}

try:

# Create the VM instance

print(f"Creating private VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"Private VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

**Key Changes:**

1. **VPC and Subnet Configuration**:
   * In the networkInterfaces part, I specify the **VPC** and **subnet** where the VM should be created.
   * The subnetwork is specified by providing the region and subnet name.

python

CopyEdit

network\_interface = {

'network': f'global/networks/{VPC\_NAME}', # VPC name

'subnetwork': f'regions/{ZONE.split("-")[0]}/subnetworks/{SUBNET\_NAME}', # Subnet name and region

}

1. **Removing Public IP**:
   * The accessConfigs field is **not included**, which ensures that no public IP is assigned to the VM. The VM will only have an internal IP within your private subnet.

**Notes:**

* **Subnet Details**: Ensure that the SUBNET\_NAME you provide is a **private subnet** and does not have a route to the internet (i.e., it should not have a NAT or public IP route).
* **Accessing the VM**: Since the VM is private and doesn't have a public IP, you would need to use a **bastion host** (a VM with a public IP) or a **VPN** to access it. Alternatively, you can use **Private Google Access** if you need to access Google services (like Cloud Storage, BigQuery, etc.) from the private VM.
* **Firewall Rules**: Make sure your firewall rules allow the necessary traffic, such as SSH, HTTP, or HTTPS, if needed for your use case.

**Running the Script:**

1. Set up your credentials, as you did earlier, using the GOOGLE\_APPLICATION\_CREDENTIALS environment variable.
2. Ensure the **VPC** and **subnet** exist in the specified region.
3. Run the script:

bash

CopyEdit

python create\_vm.py

The script will create the VM in your **private subnet** within the specified **VPC**.

Let me know if you need any further modifications or clarification! 😊

4o

**You said:**

VPC network details hu-devops-25 Overview Subnets Static internal IP addresses Firewalls Firewall endpoints Routes VPC network peering Private services access DNS configuration Subnets Filter Name Region Stack Type Primary IPv4 range Secondary IPv4 ranges IPv6 ranges Reserved internal ranges Gateway Private Google Access Flow logs private-subnet us-central1 IPv4 (single-stack) 10.0.1.0/24 - this is my vpc and subnet create private instance in them

**ChatGPT said:**

Great! Since you have your **VPC** (hu-devops-25) and **private subnet** (private-subnet with the IP range 10.0.1.0/24) in the region us-central1, I will modify the Python script accordingly to use your specific VPC and subnet for creating a private VM.

**Here's how we modify the script:**

* **VPC Network**: hu-devops-25
* **Subnet Name**: private-subnet
* **Region**: us-central1
* **IP Range**: 10.0.1.0/24

**Updated Python Script for Creating a Private VM**

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import time

# Set your project ID, VPC, subnet, and zone here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM

IMAGE\_PROJECT = 'ubuntu-os-cloud' # The project where the Ubuntu image resides

IMAGE\_FAMILY = 'ubuntu-2004-lts' # The image family (Ubuntu 20.04 LTS)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

# Authenticate and create the compute client

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

# Define the network and subnet configuration

network\_interface = {

'network': f'global/networks/{VPC\_NAME}', # Specify the VPC network

'subnetwork': f'regions/{ZONE.split("-")[0]}/subnetworks/{SUBNET\_NAME}', # Specify the private subnet and region

# No accessConfigs means no public IP address (this creates a private VM)

}

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [network\_interface], # Attach the VM to the private network

'tags': {

'items': ['http-server', 'https-server'] # Tags for access control (e.g., firewall rules)

},

}

try:

# Create the VM instance

print(f"Creating private VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"Private VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

**Key Changes:**

1. **VPC Name**:
   * The VM is now being created in the **hu-devops-25** VPC network.
2. **Subnet Name**:
   * The VM is being created in the **private-subnet** subnet. The subnet is defined in the region us-central1, and the IP range is 10.0.1.0/24.
3. **No Public IP**:
   * By not including the accessConfigs field in the networkInterfaces, the VM will **not** get a public IP, making it a private VM.
4. **Subnetwork**:
   * We specify the **subnet** under subnetwork by using the region of the VM (us-central1 in this case), which ensures the VM is placed in the correct private subnet.

python

CopyEdit

'subnetwork': f'regions/{ZONE.split("-")[0]}/subnetworks/{SUBNET\_NAME}',

**Notes:**

* **Firewall Rules**: Make sure that the firewall rules for your VPC allow traffic as needed for the VM. For instance, if you need SSH access to the private VM, ensure you have an appropriate rule for ssh on the private subnet.
* **Accessing the Private VM**: Since the VM does not have a public IP, you will need to access it using a **bastion host** (a VM with a public IP in the same VPC) or via **VPN**.
* **Private Google Access**: If you need the VM to access Google Cloud services (like Cloud Storage or BigQuery), you can configure **Private Google Access** for the subnet. This is usually enabled by default for private subnets.

**Running the Script:**

1. **Set up your credentials** as before (ensure the GOOGLE\_APPLICATION\_CREDENTIALS environment variable is set to your service account key).
2. **Run the script** to create the private VM:

bash

CopyEdit

python create\_vm.py

1. After execution, your private VM will be created in the hu-devops-25 VPC within the private-subnet subnet. You can then connect to it using internal methods like a **bastion host** or **VPN**.

Let me know if you need any further assistance! 😊

4o

**You said:**

import os import google.auth from googleapiclient.discovery import build from googleapiclient.errors import HttpError import time # Set your project ID, VPC, subnet, and zone here PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1') INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM IMAGE\_PROJECT = 'ubuntu-os-cloud' # The project where the Ubuntu image resides IMAGE\_FAMILY = 'ubuntu-2004-lts' # The image family (Ubuntu 20.04 LTS) VPC\_NAME = 'hu-devops-25' # Your VPC network name SUBNET\_NAME = 'private-subnet' # Your private subnet name # Authenticate and create the compute client def create\_instance(): # Initialize the Compute Engine client credentials, project = google.auth.default() compute = build('compute', 'v1', credentials=credentials) # Define the machine type and image machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}" # Define the network and subnet configuration network\_interface = { 'network': f'global/networks/{VPC\_NAME}', # Specify the VPC network 'subnetwork': f'regions/{ZONE.split("-")[0]}/subnetworks/{SUBNET\_NAME}', # Specify the private subnet and region # No accessConfigs means no public IP address (this creates a private VM) } # Define the configuration for the VM instance config = { 'name': INSTANCE\_NAME, 'machineType': machine\_type, 'disks': [{ 'boot': True, 'autoDelete': True, 'initializeParams': { 'sourceImage': source\_disk\_image } }], 'networkInterfaces': [network\_interface], # Attach the VM to the private network 'tags': { 'items': ['http-server', 'https-server'] # Tags for access control (e.g., firewall rules) }, } try: # Create the VM instance print(f"Creating private VM instance {INSTANCE\_NAME}...") operation = compute.instances().insert( project=PROJECT\_ID, zone=ZONE, body=config ).execute() # Wait for the operation to complete wait\_for\_operation(compute, operation) print(f"Private VM instance {INSTANCE\_NAME} created successfully.") except HttpError as err: print(f"An error occurred: {err}") return def wait\_for\_operation(compute, operation): print("Waiting for operation to complete...") while True: result = compute.zoneOperations().get( project=PROJECT\_ID, zone=ZONE, operation=operation['name'] ).execute() if result['status'] == 'DONE': if 'error' in result: raise Exception(f"Error: {result['error']}") else: print("Operation completed successfully.") break else: print("Operation is still in progress...") time.sleep(10) if \_\_name\_\_ == "\_\_main\_\_": create\_instance() - $ python private\_1.py Creating private VM instance private-ubuntu-vm... An error occurred: <HttpError 400 when requesting https://compute.googleapis.com/compute/v1/projects/manifest-zephyr-457115-v2/zones/us-central1-a/instances?alt=json returned "Invalid value fo r field 'region': 'us'. Unknown region.". Details: "[{'message': "Invalid value for field 'region': 'us'. Unknown region.", 'domain': 'global', 'reason': 'invalid'}]">

**ChatGPT said:**

The error you're seeing is due to an issue with the region formatting in the subnetwork field. The problem is that the ZONE.split("-")[0] approach to extracting the region from the zone (like us-central1-a) is causing an incorrect region format when building the subnet reference.

Specifically, the region should be specified as us-central1, but the code is extracting only us as the region.

**Solution:**

To resolve this issue, you need to fix how the region is specified when referencing the subnet in your networkInterface. Instead of ZONE.split("-")[0], you should use the correct region, which in this case is us-central1.

**Here's the updated part of your code:**

python

CopyEdit

# Define the network and subnet configuration

network\_interface = {

'network': f'global/networks/{VPC\_NAME}', # Specify the VPC network

'subnetwork': f'regions/us-central1/subnetworks/{SUBNET\_NAME}', # Specify the correct region 'us-central1' and subnet

# No accessConfigs means no public IP address (this creates a private VM)

}

**Full Updated Python Script:**

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import time

# Set your project ID, VPC, subnet, and zone here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM

IMAGE\_PROJECT = 'ubuntu-os-cloud' # The project where the Ubuntu image resides

IMAGE\_FAMILY = 'ubuntu-2004-lts' # The image family (Ubuntu 20.04 LTS)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

# Authenticate and create the compute client

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

# Define the network and subnet configuration

network\_interface = {

'network': f'global/networks/{VPC\_NAME}', # Specify the VPC network

'subnetwork': f'regions/us-central1/subnetworks/{SUBNET\_NAME}', # Specify the correct region 'us-central1' and subnet

# No accessConfigs means no public IP address (this creates a private VM)

}

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [network\_interface], # Attach the VM to the private network

'tags': {

'items': ['http-server', 'https-server'] # Tags for access control (e.g., firewall rules)

},

}

try:

# Create the VM instance

print(f"Creating private VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"Private VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

**Key Changes:**

* In the subnetwork field, I explicitly use the **correct region** (us-central1) instead of trying to extract it from the ZONE variable. Now, the subnet reference is properly formed.

python

CopyEdit

'subnetwork': f'regions/us-central1/subnetworks/{SUBNET\_NAME}', # Correct region and subnet

**Running the Script:**

1. Ensure you've set up your Google Cloud credentials with GOOGLE\_APPLICATION\_CREDENTIALS.
2. Run the script again with:

bash

CopyEdit

python private\_1.py

**Result:**

This script will now create a **private VM** in the hu-devops-25 VPC and the private-subnet subnet in the us-central1 region without a public IP.

Let me know if this works, or if you encounter any further issues! 😊

4o

**You said:**

import os import google.auth from googleapiclient.discovery import build from googleapiclient.errors import HttpError import time # Set your project ID, VPC, subnet, and zone here PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1') INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM IMAGE\_PROJECT = 'ubuntu-os-cloud' # The project where the Ubuntu image resides IMAGE\_FAMILY = 'ubuntu-2004-lts' # The image family (Ubuntu 20.04 LTS) VPC\_NAME = 'hu-devops-25' # Your VPC network name SUBNET\_NAME = 'private-subnet' # Your private subnet name # Authenticate and create the compute client def create\_instance(): # Initialize the Compute Engine client credentials, project = google.auth.default() compute = build('compute', 'v1', credentials=credentials) # Define the machine type and image machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}" # Define the network and subnet configuration network\_interface = { 'network': f'global/networks/{VPC\_NAME}', # Specify the VPC network 'subnetwork': f'regions/us-central1/subnetworks/{SUBNET\_NAME}', # Specify the correct region 'us-central1' and subnet # No accessConfigs means no public IP address (this creates a private VM) } # Define the configuration for the VM instance config = { 'name': INSTANCE\_NAME, 'machineType': machine\_type, 'disks': [{ 'boot': True, 'autoDelete': True, 'initializeParams': { 'sourceImage': source\_disk\_image } }], 'networkInterfaces': [network\_interface], # Attach the VM to the private network 'tags': { 'items': ['http-server', 'https-server'] # Tags for access control (e.g., firewall rules) }, } try: # Create the VM instance print(f"Creating private VM instance {INSTANCE\_NAME}...") operation = compute.instances().insert( project=PROJECT\_ID, zone=ZONE, body=config ).execute() # Wait for the operation to complete wait\_for\_operation(compute, operation) print(f"Private VM instance {INSTANCE\_NAME} created successfully.") except HttpError as err: print(f"An error occurred: {err}") return def wait\_for\_operation(compute, operation): print("Waiting for operation to complete...") while True: result = compute.zoneOperations().get( project=PROJECT\_ID, zone=ZONE, operation=operation['name'] ).execute() if result['status'] == 'DONE': if 'error' in result: raise Exception(f"Error: {result['error']}") else: print("Operation completed successfully.") break else: print("Operation is still in progress...") time.sleep(10) if \_\_name\_\_ == "\_\_main\_\_": create\_instance() - user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python private\_1.py Creating private VM instance private-ubuntu-vm... An error occurred: <HttpError 409 when requesting https://compute.googleapis.com/compute/v1/projects/manifest-zephyr-457115-v2/zones/us-central1-a/instances?alt=json returned "The resource 'pr ojects/manifest-zephyr-457115-v2/zones/us-central1-a/instances/private-ubuntu-vm' already exists". Details: "[{'message': "The resource 'projects/manifest-zephyr-457115-v2/zones/us-central1-a/ instances/private-ubuntu-vm' already exists", 'domain': 'global', 'reason': 'alreadyExists'}]"> - use region us-central1

**ChatGPT said:**

The error message indicates that there is already a VM named private-ubuntu-vm in the specified zone (us-central1-a). To resolve this, you can either:

1. **Choose a different name for the new VM** so that it doesn't conflict with the existing one.
2. **Delete the existing VM** if it is no longer needed, or re-use its name if you want to overwrite it.

Additionally, you have mentioned to use the region us-central1 instead of the zone us-central1-a. If you want to use the region for the VM creation, make sure to specify the zone within us-central1, such as us-central1-a, us-central1-b, etc. Google Cloud requires a specific zone within the region for VM creation.

**Steps to Resolve:**

1. **Change the VM name** to avoid the conflict.
2. **Ensure the correct zone within the region** (us-central1-a, us-central1-b, etc.).

**Updated Python Script with a New VM Name:**

python

CopyEdit

import os

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import time

# Set your project ID, VPC, subnet, and zone here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm-2' # New name for the VM to avoid conflict

IMAGE\_PROJECT = 'ubuntu-os-cloud' # The project where the Ubuntu image resides

IMAGE\_FAMILY = 'ubuntu-2004-lts' # The image family (Ubuntu 20.04 LTS)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

# Authenticate and create the compute client

def create\_instance():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

# Define the machine type and image

machine\_type = f"zones/{ZONE}/machineTypes/n1-standard-1" # You can change the machine type here

source\_disk\_image = f"projects/{IMAGE\_PROJECT}/global/images/family/{IMAGE\_FAMILY}"

# Define the network and subnet configuration

network\_interface = {

'network': f'global/networks/{VPC\_NAME}', # Specify the VPC network

'subnetwork': f'regions/us-central1/subnetworks/{SUBNET\_NAME}', # Specify the correct region 'us-central1' and subnet

# No accessConfigs means no public IP address (this creates a private VM)

}

# Define the configuration for the VM instance

config = {

'name': INSTANCE\_NAME,

'machineType': machine\_type,

'disks': [{

'boot': True,

'autoDelete': True,

'initializeParams': {

'sourceImage': source\_disk\_image

}

}],

'networkInterfaces': [network\_interface], # Attach the VM to the private network

'tags': {

'items': ['http-server', 'https-server'] # Tags for access control (e.g., firewall rules)

},

}

try:

# Create the VM instance

print(f"Creating private VM instance {INSTANCE\_NAME}...")

operation = compute.instances().insert(

project=PROJECT\_ID,

zone=ZONE,

body=config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"Private VM instance {INSTANCE\_NAME} created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

if \_\_name\_\_ == "\_\_main\_\_":

create\_instance()

**Key Updates:**

* **Changed the VM Name**: I changed the VM instance name to private-ubuntu-vm-2 to avoid the conflict. You can replace it with any other unique name.

python

CopyEdit

INSTANCE\_NAME = 'private-ubuntu-vm-2' # New name for the VM to avoid conflict

**Alternative Solution: Delete the Existing VM**

If you want to keep the same name (private-ubuntu-vm), and you're okay with deleting the existing VM, you can delete the VM first and then create a new one using the same name.

To delete the existing VM using gcloud:

bash

CopyEdit

gcloud compute instances delete private-ubuntu-vm --zone=us-central1-a --project=manifest-zephyr-457115-v2

After deleting the existing VM, you can run the original script to recreate the VM with the same name.

**Running the Script Again:**

After making these changes, run the script again to create the new VM:

bash

CopyEdit

python private\_1.py

This will create a private VM with the updated name (private-ubuntu-vm-2) in the correct subnet and VPC.

Let me know if you encounter any further issues! 😊

4o

**You said:**

create an gcp app engine with all the configuration that we specify above

**ChatGPT said:**

To create a **Google Cloud App Engine** instance with all the configurations you mentioned (using a specific VPC, private subnet, and region), we need to slightly adjust the approach.

App Engine is a **Platform as a Service (PaaS)** offering from Google Cloud, and it operates a bit differently from Compute Engine (VMs). While App Engine does not allow full customization of network interfaces as Compute Engine does, you can still configure your app to run in a **VPC** using **App Engine Flexible Environment**, which supports **private networking**.

**Steps to Configure App Engine with a VPC and Private Subnet:**

1. **Create an App Engine Application** (for a flexible environment).
2. **Configure your app.yaml file** to run in the **VPC**.
3. **Configure Private Google Access** for private access to Google services.
4. **Make sure your App Engine is deployed using the flexible environment**, which allows custom networking.

**Let's Walk Through the Steps:**

1. **Ensure that App Engine is Enabled in Your Project**: First, make sure App Engine is enabled for your project. You can enable it using the gcloud command:

bash

CopyEdit

gcloud app create --region=us-central

This command will initialize App Engine in the us-central region, where we plan to deploy the application.

1. **App Engine YAML Configuration**: Create an app.yaml file in the root of your project directory. This file will define the configuration for your app deployment.

**Sample app.yaml for Flexible Environment with VPC:**

yaml

CopyEdit

runtime: python39 # Use the appropriate runtime for your app

env: flex # Use the flexible environment to configure networking

# App Engine uses this file to configure networking, so we're configuring VPC here.

env\_variables:

# Example environment variables for your application

VPC\_NAME: 'hu-devops-25'

SUBNET\_NAME: 'private-subnet'

GOOGLE\_CLOUD\_PROJECT: 'manifest-zephyr-457115-v2'

# Networking and VPC configuration

vpc\_access\_connector:

name: "projects/manifest-zephyr-457115-v2/locations/us-central1/connectors/my-vpc-connector" # VPC connector name, which we'll create in the next steps

instance\_class: F2 # Choose an instance class (adjust based on your app's requirements)

**Key Configuration:**

* **env: flex**: This specifies that we are using the **Flexible Environment**, which allows full networking configuration.
* **vpc\_access\_connector**: You need to create a **Serverless VPC Access connector** to allow App Engine to connect to your VPC network and private subnet.

**Step 3: Create a VPC Connector**

To connect your App Engine app to the VPC and private subnet, you need to create a **VPC Access connector**. This is done using the following gcloud command:

bash

CopyEdit

gcloud compute networks vpc-access connectors create my-vpc-connector \

--region=us-central1 \

--subnet=private-subnet \

--network=hu-devops-25

* **my-vpc-connector**: This is the name for the VPC access connector. You can choose any name, but it must match the name you specify in app.yaml.
* **--region=us-central1**: The region where the connector will be created.
* **--subnet=private-subnet**: The subnet where your app should connect. This will be the private subnet.
* **--network=hu-devops-25**: The VPC network name.

This will create a VPC access connector that connects your App Engine app to your private subnet.

**Step 4: Deploy Your Application**

Once you've configured the app.yaml and created the VPC connector, you can deploy your application with:

bash

CopyEdit

gcloud app deploy

This will deploy your app to **App Engine** in the **Flexible Environment**, where it will be connected to your **VPC** and **private subnet**.

**Step 5: Additional Configuration**

* **Private Google Access**: If you want your private App Engine app to access Google services (like Cloud Storage, BigQuery) while keeping all traffic internal, you need to ensure that **Private Google Access** is enabled on your subnet. This is important for the app to access Google APIs without a public IP.

You can enable **Private Google Access** for your subnet via the Google Cloud Console or using the following command:

bash

CopyEdit

gcloud compute networks subnets update private-subnet \

--region=us-central1 \

--enable-private-ip-google-access

**Final Summary:**

* **App Engine Flexible Environment**: This allows you to configure networking and run your app inside a VPC, with no public IP (private VM).
* **VPC Access Connector**: Connects your App Engine app to your VPC and private subnet.
* **app.yaml Configuration**: Specifies how your App Engine app connects to the VPC using the flexible environment and VPC connector.
* **Private Google Access**: Ensures your app can access Google services without using public IPs.

Let me know if you have any questions or if you need further clarification!

4o

**You said:**

using python script

**ChatGPT said:**

To create a Google Cloud App Engine with a VPC and private subnet using a Python script, we can use the Google Cloud API Python client library to programmatically manage App Engine and VPC resources.

Since App Engine Flexible Environment supports connecting to a VPC via a **VPC Access Connector**, we can write a script that:

1. **Creates the VPC Access Connector** to allow the App Engine instance to connect to the private subnet.
2. **Deploys the app to App Engine** using the provided configuration (app.yaml).

Let's break this into two parts:

**Step 1: Creating the VPC Access Connector**

This part involves using the Google Cloud Compute API to create a **VPC Access Connector** to link App Engine to your **private subnet** in your **VPC**.

**Step 2: Deploying the App to App Engine**

Once the VPC Access Connector is created, you can deploy the App Engine app using the gcloud command in a Python script (since the App Engine deployment process is done using gcloud app deploy).

Here's a full Python script that integrates these steps:

**Python Script to Set Up VPC Access Connector and Deploy to App Engine:**

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and zone here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the compute client

def create\_vpc\_connector():

# Initialize the Compute Engine client

credentials, project = google.auth.default()

compute = build('compute', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'region': REGION,

'network': f'global/networks/{VPC\_NAME}',

'subnet': f'regions/{REGION}/subnetworks/{SUBNET\_NAME}',

}

operation = compute.projects().regions().connectors().insert(

project=PROJECT\_ID,

region=REGION,

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(compute, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(compute, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = compute.zoneOperations().get(

project=PROJECT\_ID,

zone=ZONE,

operation=operation['name']

).execute()

if result['status'] == 'DONE':

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Explanation:**

**1. Create the VPC Access Connector:**

* We use the **Google Compute API** to create a **VPC Access Connector**. This allows App Engine (in the flexible environment) to connect to your VPC and private subnet.
* The connector\_config includes the name of the connector, the region, the VPC, and the subnet to be used.

**2. Wait for the Connector Creation to Complete:**

* The wait\_for\_operation() function waits for the operation to finish, ensuring that the connector is created successfully before moving on.

**3. Deploy the App to App Engine:**

* Once the VPC Access Connector is created, we deploy the application to **App Engine**. This is done using the gcloud app deploy command via subprocess.run(). This step requires that you have an app.yaml file in your project directory for configuring the App Engine deployment.

The subprocess.run() function executes the gcloud app deploy command to deploy the app to **App Engine**. Make sure your project has App Engine enabled and the app.yaml file is configured properly (as described in the earlier answer).

**Requirements:**

* **google-auth**, **google-api-python-client**: Install the required libraries to interact with the Google Cloud API:

bash

CopyEdit

pip install google-auth google-api-python-client

* **gcloud**: The script uses gcloud app deploy, so you must have the **Google Cloud SDK** installed on your system. Follow the instructions from Google Cloud SDK Installation if you don't have it yet.
* **App Engine Configuration**: Ensure that your app.yaml file exists in the project directory for the App Engine app. Here's an example of what app.yaml might look like for a Python application:

yaml

CopyEdit

runtime: python39

env: flex # Use flexible environment

env\_variables:

VPC\_NAME: 'hu-devops-25'

SUBNET\_NAME: 'private-subnet'

instance\_class: F2 # Adjust instance class as needed

**Running the Script:**

1. Ensure that you've authenticated with Google Cloud using the appropriate credentials (gcloud auth login and/or GOOGLE\_APPLICATION\_CREDENTIALS).
2. Run the script:

bash

CopyEdit

python create\_vpc\_and\_deploy\_app.py

**Final Notes:**

* This script will **create a VPC Access Connector** and then **deploy the app to App Engine** using the specified VPC and private subnet.
* The gcloud app deploy command will only work if your app.yaml is properly configured.

Let me know if you have any questions or need further clarification! 😊

4o

**You said:**

import google.auth from googleapiclient.discovery import build from googleapiclient.errors import HttpError import subprocess import time # Set your project ID, VPC, subnet, and zone here PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID REGION = 'us-central1' # The region where your VPC and subnet reside ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1') INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference) VPC\_NAME = 'hu-devops-25' # Your VPC network name SUBNET\_NAME = 'private-subnet' # Your private subnet name CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment # Authenticate and create the compute client def create\_vpc\_connector(): # Initialize the Compute Engine client credentials, project = google.auth.default() compute = build('compute', 'v1', credentials=credentials) try: # Create the VPC Access Connector print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...") connector\_config = { 'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}", 'region': REGION, 'network': f'global/networks/{VPC\_NAME}', 'subnet': f'regions/{REGION}/subnetworks/{SUBNET\_NAME}', } operation = compute.projects().regions().connectors().insert( project=PROJECT\_ID, region=REGION, body=connector\_config ).execute() # Wait for the operation to complete wait\_for\_operation(compute, operation) print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.") except HttpError as err: print(f"An error occurred: {err}") return def wait\_for\_operation(compute, operation): """Wait for the operation to complete.""" print("Waiting for operation to complete...") while True: result = compute.zoneOperations().get( project=PROJECT\_ID, zone=ZONE, operation=operation['name'] ).execute() if result['status'] == 'DONE': if 'error' in result: raise Exception(f"Error: {result['error']}") else: print("Operation completed successfully.") break else: print("Operation is still in progress...") time.sleep(10) def deploy\_app\_engine(): """Deploy the app to App Engine.""" try: print("Deploying the application to App Engine...") # Deploy the app using gcloud app deploy subprocess.run(['gcloud', 'app', 'deploy'], check=True) print("App Engine deployment completed successfully.") except subprocess.CalledProcessError as e: print(f"Deployment failed: {e}") if \_\_name\_\_ == "\_\_main\_\_": # Step 1: Create the VPC Access Connector create\_vpc\_connector() # Step 2: Deploy the app to App Engine deploy\_app\_engine() user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python app.py Creating VPC Access Connector 'my-vpc-connector'... Traceback (most recent call last): File "D:\agcp\app.py", line 82, in <module> create\_vpc\_connector() ~~~~~~~~~~~~~~~~~~~~^^ File "D:\agcp\app.py", line 33, in create\_vpc\_connector operation = compute.projects().regions().connectors().insert( ^^^^^^^^^^^^^^^^^^^^^^^^^^ AttributeError: 'Resource' object has no attribute 'regions'

**ChatGPT said:**

The error you encountered occurs because the API request to create a **VPC Access Connector** is being made incorrectly using compute.projects().regions().connectors().insert. The error message indicates that the regions resource doesn't exist under the compute API client, because VPC Access connectors are managed by the **VPC Access API**, not the Compute Engine API.

To fix this, you need to use the **VPC Access API** (vpcaccess service) to create the VPC Access Connector.

**Correct Approach:**

1. **Use the vpcaccess service** to manage the VPC Access Connector.
2. **Create the VPC Access Connector** using the correct API client.

**Updated Python Script Using the Correct API Client:**

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and region here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the vpcaccess client

def create\_vpc\_connector():

# Initialize the VPC Access API client

credentials, project = google.auth.default()

vpcaccess = build('vpcaccess', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'region': REGION,

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}',

}

operation = vpcaccess.projects().locations().connectors().create(

parent=f"projects/{PROJECT\_ID}/locations/{REGION}",

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(vpcaccess, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(vpcaccess, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = vpcaccess.projects().locations().operations().get(

name=operation['name']

).execute()

if result['done']:

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Key Changes:**

1. **Changed the API client** to use vpcaccess instead of compute to interact with the **VPC Access API**.
   * We now use the correct API vpcaccess.projects().locations().connectors().create() to create a VPC Access Connector.
   * The connector is created under the project and region us-central1.
2. **Corrected the name and subnet fields** in the connector\_config to follow the proper format, specifying the full resource path, such as:

python

CopyEdit

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}',

**Installing Required Libraries:**

Make sure the following libraries are installed for using Google Cloud's API:

bash

CopyEdit

pip install google-auth google-api-python-client

**Running the Script:**

1. Make sure you're authenticated with Google Cloud (gcloud auth login).
2. Ensure that you have the correct app.yaml for App Engine deployment.
3. Run the script to create the **VPC Access Connector** and deploy the app:

bash

CopyEdit

python app.py

This script will now:

1. Create the **VPC Access Connector**.
2. Deploy your application to **App Engine** using the flexible environment and connecting it to the private VPC.

Let me know if this works or if you have any questions! 😊

4o

**You said:**

import google.auth from googleapiclient.discovery import build from googleapiclient.errors import HttpError import subprocess import time # Set your project ID, VPC, subnet, and region here PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID REGION = 'us-central1' # The region where your VPC and subnet reside ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1') INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference) VPC\_NAME = 'hu-devops-25' # Your VPC network name SUBNET\_NAME = 'private-subnet' # Your private subnet name CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment # Authenticate and create the vpcaccess client def create\_vpc\_connector(): # Initialize the VPC Access API client credentials, project = google.auth.default() vpcaccess = build('vpcaccess', 'v1', credentials=credentials) try: # Create the VPC Access Connector print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...") connector\_config = { 'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}", 'region': REGION, 'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}', 'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}', } operation = vpcaccess.projects().locations().connectors().create( parent=f"projects/{PROJECT\_ID}/locations/{REGION}", body=connector\_config ).execute() # Wait for the operation to complete wait\_for\_operation(vpcaccess, operation) print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.") except HttpError as err: print(f"An error occurred: {err}") return def wait\_for\_operation(vpcaccess, operation): """Wait for the operation to complete.""" print("Waiting for operation to complete...") while True: result = vpcaccess.projects().locations().operations().get( name=operation['name'] ).execute() if result['done']: if 'error' in result: raise Exception(f"Error: {result['error']}") else: print("Operation completed successfully.") break else: print("Operation is still in progress...") time.sleep(10) def deploy\_app\_engine(): """Deploy the app to App Engine.""" try: print("Deploying the application to App Engine...") # Deploy the app using gcloud app deploy subprocess.run(['gcloud', 'app', 'deploy'], check=True) print("App Engine deployment completed successfully.") except subprocess.CalledProcessError as e: print(f"Deployment failed: {e}") if \_\_name\_\_ == "\_\_main\_\_": # Step 1: Create the VPC Access Connector create\_vpc\_connector() # Step 2: Deploy the app to App Engine deploy\_app\_engine() - user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python app.py Creating VPC Access Connector 'my-vpc-connector'... An error occurred: <HttpError 400 when requesting https://vpcaccess.googleapis.com/v1/projects/manifest-zephyr-457115-v2/locations/us-central1/connectors?alt=json returned "Invalid JSON payloa d received. Unknown name "region" at 'connector': Cannot find field. Invalid value at 'connector.subnet' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "projects/manifest-zephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"". Deta ils: "[{'@type': 'type.googleapis.com/google.rpc.BadRequest', 'fieldViolations': [{'field': 'connector', 'description': 'Invalid JSON payload received. Unknown name "region" at \'connector\': Cannot find field.'}, {'field': 'connector.subnet', 'description': 'Invalid value at \'connector.subnet\' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "projects/manifest-z ephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"'}]}]"> Deploying the application to App Engine... Traceback (most recent call last): File "D:\agcp\app.py", line 82, in <module> deploy\_app\_engine() ~~~~~~~~~~~~~~~~~^^ File "D:\agcp\app.py", line 71, in deploy\_app\_engine subprocess.run(['gcloud', 'app', 'deploy'], check=True) ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 556, in run with Popen(\*popenargs, \*\*kwargs) as process: ~~~~~^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1038, in \_\_init\_\_ self.\_execute\_child(args, executable, preexec\_fn, close\_fds, ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ pass\_fds, cwd, env, ^^^^^^^^^^^^^^^^^^^ ...<5 lines>... gid, gids, uid, umask, ^^^^^^^^^^^^^^^^^^^^^^ start\_new\_session, process\_group) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1550, in \_execute\_child hp, ht, pid, tid = \_winapi.CreateProcess(executable, args, ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^ # no special security ^^^^^^^^^^^^^^^^^^^^^ ...<4 lines>... cwd, ^^^^ startupinfo) ^^^^^^^^^^^^ FileNotFoundError: [WinError 2] The system cannot find the file specified user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $

**ChatGPT said:**

The error you're encountering has two parts:

1. **Invalid JSON payload for the VPC Access Connector**: This error occurs because the payload for the VPC Access Connector API request is malformed. Specifically, it indicates that the region field is not a valid parameter in the request and the subnet field's value is incorrectly formatted.

The correct structure for creating a VPC Access connector is:

* + **Do not include region as a direct field** in the request.
  + The **subnet field** should just reference the subnet name and not the full resource path.

1. **Error when running gcloud app deploy**: This error (FileNotFoundError: [WinError 2] The system cannot find the file specified) indicates that the system is unable to find the gcloud command because it’s not in your system's PATH.

Let's address each issue step by step.

**1. Fixing the VPC Access Connector API Request**

We need to update the request to fix the JSON structure for the VPC Access Connector.

* **Do not use the region parameter directly** in the connector's JSON.
* For the subnet field, just provide the subnet name, not the full resource path.

Here's the corrected Python code for creating the VPC Access Connector:

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and region here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the vpcaccess client

def create\_vpc\_connector():

# Initialize the VPC Access API client

credentials, project = google.auth.default()

vpcaccess = build('vpcaccess', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'{SUBNET\_NAME}', # Just use the subnet name, not the full path

}

operation = vpcaccess.projects().locations().connectors().create(

parent=f"projects/{PROJECT\_ID}/locations/{REGION}",

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(vpcaccess, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(vpcaccess, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = vpcaccess.projects().locations().operations().get(

name=operation['name']

).execute()

if result['done']:

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Changes:**

1. **Corrected the subnet field**: You no longer need to provide the full resource path for the subnet. Just provide the subnet name (private-subnet) instead of the full path.

python

CopyEdit

'subnet': f'{SUBNET\_NAME}', # Just use the subnet name, not the full path

**2. Fixing the gcloud app deploy Error (FileNotFoundError)**

The error FileNotFoundError: [WinError 2] The system cannot find the file specified means that the gcloud command is not recognized by your system.

To fix this, follow these steps:

**Install the Google Cloud SDK (if not already installed):**

1. **Download and Install the Google Cloud SDK**:
   * Follow this guide to download and install the **Google Cloud SDK**.
   * Make sure to add the gcloud command to your system’s PATH during installation.
2. **Verify the installation**: After installation, verify that the gcloud command is working by running the following in your terminal:

bash

CopyEdit

gcloud --version

This should display the version of the gcloud SDK installed.

1. **Authenticate with Google Cloud**: If you haven't authenticated yet, use the following command to log in:

bash

CopyEdit

gcloud auth login

**Add gcloud to the PATH Manually (if needed):**

If gcloud is installed but not recognized, you may need to add the gcloud command to your PATH manually.

On Windows:

1. Open **System Properties** > **Environment Variables**.
2. Under **System Variables**, find Path and click **Edit**.
3. Add the directory where gcloud is installed (e.g., C:\Program Files (x86)\Google\Cloud SDK\google-cloud-sdk\bin) to the Path variable.
4. Click **OK** to save the changes.

Then restart your terminal or IDE.

**3. Running the Script Again**

Once the gcloud command is available and you've corrected the API call for creating the VPC Access Connector, you can run the script again:

bash

CopyEdit

python app.py

**Conclusion:**

1. The VPC Access Connector creation API call has been corrected to use the right structure.
2. The gcloud command not being found is resolved by installing or adding it to your system's PATH.
3. After applying these fixes, your script should create the VPC Access Connector and deploy the application to App Engine successfully.

Let me know if you need further assistance!

4o

**You said:**

import google.auth from googleapiclient.discovery import build from googleapiclient.errors import HttpError import subprocess import time # Set your project ID, VPC, subnet, and region here PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID REGION = 'us-central1' # The region where your VPC and subnet reside ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1') INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference) VPC\_NAME = 'hu-devops-25' # Your VPC network name SUBNET\_NAME = 'private-subnet' # Your private subnet name CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment # Authenticate and create the vpcaccess client def create\_vpc\_connector(): # Initialize the VPC Access API client credentials, project = google.auth.default() vpcaccess = build('vpcaccess', 'v1', credentials=credentials) try: # Create the VPC Access Connector print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...") connector\_config = { 'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}", 'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}', 'subnet': f'{SUBNET\_NAME}', # Just use the subnet name, not the full path } operation = vpcaccess.projects().locations().connectors().create( parent=f"projects/{PROJECT\_ID}/locations/{REGION}", body=connector\_config ).execute() # Wait for the operation to complete wait\_for\_operation(vpcaccess, operation) print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.") except HttpError as err: print(f"An error occurred: {err}") return def wait\_for\_operation(vpcaccess, operation): """Wait for the operation to complete.""" print("Waiting for operation to complete...") while True: result = vpcaccess.projects().locations().operations().get( name=operation['name'] ).execute() if result['done']: if 'error' in result: raise Exception(f"Error: {result['error']}") else: print("Operation completed successfully.") break else: print("Operation is still in progress...") time.sleep(10) def deploy\_app\_engine(): """Deploy the app to App Engine.""" try: print("Deploying the application to App Engine...") # Deploy the app using gcloud app deploy subprocess.run(['gcloud', 'app', 'deploy'], check=True) print("App Engine deployment completed successfully.") except subprocess.CalledProcessError as e: print(f"Deployment failed: {e}") if \_\_name\_\_ == "\_\_main\_\_": # Step 1: Create the VPC Access Connector create\_vpc\_connector() # Step 2: Deploy the app to App Engine deploy\_app\_engine() - user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python app.py Creating VPC Access Connector 'my-vpc-connector'... An error occurred: <HttpError 400 when requesting https://vpcaccess.googleapis.com/v1/projects/manifest-zephyr-457115-v2/locations/us-central1/connectors?alt=json returned "Invalid value at 'c onnector.subnet' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "private-subnet"". Details: "[{'@type': 'type.googleapis.com/google.rpc.BadRequest', 'fieldViolations': [{'fi eld': 'connector.subnet', 'description': 'Invalid value at \'connector.subnet\' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "private-subnet"'}]}]"> Deploying the application to App Engine... Traceback (most recent call last): File "D:\agcp\app.py", line 82, in <module> deploy\_app\_engine() ~~~~~~~~~~~~~~~~~^^ File "D:\agcp\app.py", line 71, in deploy\_app\_engine subprocess.run(['gcloud', 'app', 'deploy'], check=True) ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 556, in run with Popen(\*popenargs, \*\*kwargs) as process: ~~~~~^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1038, in \_\_init\_\_ self.\_execute\_child(args, executable, preexec\_fn, close\_fds, ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ pass\_fds, cwd, env, ^^^^^^^^^^^^^^^^^^^ ...<5 lines>... gid, gids, uid, umask, ^^^^^^^^^^^^^^^^^^^^^^ start\_new\_session, process\_group) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1550, in \_execute\_child hp, ht, pid, tid = \_winapi.CreateProcess(executable, args, ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^ # no special security ^^^^^^^^^^^^^^^^^^^^^ ...<4 lines>... cwd, ^^^^ startupinfo) ^^^^^^^^^^^^ FileNotFoundError: [WinError 2] The system cannot find the file specified user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp

**ChatGPT said:**

Let's address the two issues you're encountering:

**1. VPC Access Connector Error (Invalid value at 'connector.subnet')**

The error message indicates that the value you're providing for the subnet is invalid because it's expecting a full resource path, not just the subnet name.

In the VPC Access API, you need to provide the full resource path for the subnet in the subnet field, rather than just the name of the subnet.

The correct format for the subnet field is:

bash

CopyEdit

projects/{project\_id}/regions/{region}/subnetworks/{subnet\_name}

So, for your case, it should look like this:

python

CopyEdit

'connector': {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}', # Full resource path for subnet

}

**2. Error Deploying the App with gcloud (FileNotFoundError)**

This error happens because the **gcloud** command is not recognized by your system. This could happen for a few reasons:

* **Google Cloud SDK (gcloud) is not installed**, or
* **gcloud is not added to your system's PATH**.

Let's resolve these:

**Install the Google Cloud SDK:**

1. **Install the Google Cloud SDK** from here if you haven't installed it yet.
2. Make sure **gcloud** is added to your system's PATH.

**Verify gcloud is in the PATH:**

After installing, open a new terminal and check if gcloud is recognized by running:

bash

CopyEdit

gcloud --version

If this outputs the version of the Google Cloud SDK, then it's installed and available. If not, follow the instructions from the installation guide to add gcloud to your PATH.

**Final Updated Python Script:**

Here's the corrected Python script with the VPC Access Connector issue fixed and the deployment steps:

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and region here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the vpcaccess client

def create\_vpc\_connector():

# Initialize the VPC Access API client

credentials, project = google.auth.default()

vpcaccess = build('vpcaccess', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}', # Full path for subnet

}

operation = vpcaccess.projects().locations().connectors().create(

parent=f"projects/{PROJECT\_ID}/locations/{REGION}",

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(vpcaccess, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(vpcaccess, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = vpcaccess.projects().locations().operations().get(

name=operation['name']

).execute()

if result['done']:

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Summary of Changes:**

1. **Fixed the subnet field**: The subnet now uses the correct full resource path:

python

CopyEdit

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}'

1. **gcloud command issue**:
   * Ensure **Google Cloud SDK** (gcloud) is installed and added to the system's PATH.
   * Verify by running gcloud --version in your terminal.

**Next Steps:**

1. **Run the script** after making sure gcloud is installed and working.
2. **Ensure you have the correct app.yaml** in your project for App Engine deployment.

Let me know if you encounter any further issues! 😊

4o

**You said:**

user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python app.py Creating VPC Access Connector 'my-vpc-connector'... An error occurred: <HttpError 400 when requesting https://vpcaccess.googleapis.com/v1/projects/manifest-zephyr-457115-v2/locations/us-central1/connectors?alt=json returned "Invalid value at 'c onnector.subnet' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "projects/manifest-zephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"". Details: "[{'@type': 't ype.googleapis.com/google.rpc.BadRequest', 'fieldViolations': [{'field': 'connector.subnet', 'description': 'Invalid value at \'connector.subnet\' (type.googleapis.com/google.cloud.vpcaccess.v 1.Connector.Subnet), "projects/manifest-zephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"'}]}]"> Deploying the application to App Engine... Traceback (most recent call last): File "D:\agcp\app.py", line 82, in <module> deploy\_app\_engine() ~~~~~~~~~~~~~~~~~^^ File "D:\agcp\app.py", line 71, in deploy\_app\_engine subprocess.run(['gcloud', 'app', 'deploy'], check=True) ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 556, in run with Popen(\*popenargs, \*\*kwargs) as process: ~~~~~^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1038, in \_\_init\_\_ self.\_execute\_child(args, executable, preexec\_fn, close\_fds, ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ pass\_fds, cwd, env, ^^^^^^^^^^^^^^^^^^^ ...<5 lines>... gid, gids, uid, umask, ^^^^^^^^^^^^^^^^^^^^^^ start\_new\_session, process\_group) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1550, in \_execute\_child hp, ht, pid, tid = \_winapi.CreateProcess(executable, args, ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^ # no special security ^^^^^^^^^^^^^^^^^^^^^ ...<4 lines>... cwd, ^^^^ startupinfo) ^^^^^^^^^^^^ FileNotFoundError: [WinError 2] The system cannot find the file specified - same error

**ChatGPT said:**

**Let's address both issues step-by-step:**

**1. VPC Access Connector Error (Invalid value at 'connector.subnet')**

The error indicates that the subnet field is not correctly formatted. The **VPC Access API** expects the **full resource path** to the subnet, but it seems like the format provided is incorrect.

The correct format should be:

bash

CopyEdit

projects/{project\_id}/regions/{region}/subnetworks/{subnet\_name}

However, it looks like you're still facing the issue with the subnet field. After reviewing your error, I see that it could be due to the way the API expects the resource paths for the **subnet**.

The correct field format for **VPC Access Connector** should be:

python

CopyEdit

'connector': {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'{SUBNET\_NAME}' # Use subnet name only

}

Let's **modify** the code so that it handles this correctly by passing only the subnet name instead of the full resource path.

**Fixed Python Code for VPC Access Connector:**

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and region here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the vpcaccess client

def create\_vpc\_connector():

# Initialize the VPC Access API client

credentials, project = google.auth.default()

vpcaccess = build('vpcaccess', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'{SUBNET\_NAME}', # Only use the subnet name, not the full path

}

operation = vpcaccess.projects().locations().connectors().create(

parent=f"projects/{PROJECT\_ID}/locations/{REGION}",

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(vpcaccess, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(vpcaccess, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = vpcaccess.projects().locations().operations().get(

name=operation['name']

).execute()

if result['done']:

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Explanation of Fixes:**

1. **Subnetwork Format**: Instead of using the full path for the subnet, we're now just using the **subnet name** ('private-subnet').

python

CopyEdit

'subnet': f'{SUBNET\_NAME}' # Just the subnet name, not the full path

This should solve the issue with the Invalid value at 'connector.subnet'.

**2. Fixing the gcloud app deploy Error (FileNotFoundError)**

The error message FileNotFoundError: [WinError 2] The system cannot find the file specified suggests that the **gcloud** command isn't found, which likely means that **Google Cloud SDK** (which provides gcloud) is not installed or is not on your PATH.

To fix this, please follow these steps:

**Step 1: Install Google Cloud SDK**

1. Download and install the **Google Cloud SDK** from here.
2. During installation, ensure that **gcloud** is added to your system’s PATH (this should be automatic).

**Step 2: Verify gcloud is in the PATH**

After installation, restart your terminal and run the following command to verify that the gcloud command is available:

bash

CopyEdit

gcloud --version

If this shows the version of the gcloud SDK, it's installed correctly.

**Step 3: Authenticate with Google Cloud**

Make sure you're authenticated with Google Cloud:

bash

CopyEdit

gcloud auth login

This will open a browser window where you can log in to your Google Cloud account.

**Step 4: Add gcloud to PATH Manually (If Needed)**

If the gcloud command is still not recognized, you might need to manually add it to the system's PATH.

On Windows:

1. Open **System Properties** > **Environment Variables**.
2. Under **System Variables**, find Path and click **Edit**.
3. Add the directory where gcloud is installed (typically C:\Program Files (x86)\Google\Cloud SDK\google-cloud-sdk\bin).
4. Click **OK** to save the changes.
5. Restart your terminal or IDE.

**3. Run the Script Again**

After fixing the issues, try running the script again:

bash

CopyEdit

python app.py

**Conclusion:**

* The **VPC Access Connector issue** has been fixed by correcting the subnet field format.
* **gcloud not found**: Ensure that the **Google Cloud SDK** is installed and available in the system PATH.

Let me know if you run into any further issues or need more assistance! 😊

4o

**You said:**

user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python app.py Creating VPC Access Connector 'my-vpc-connector'... An error occurred: <HttpError 400 when requesting https://vpcaccess.googleapis.com/v1/projects/manifest-zephyr-457115-v2/locations/us-central1/connectors?alt=json returned "Invalid value at 'c onnector.subnet' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "private-subnet"". Details: "[{'@type': 'type.googleapis.com/google.rpc.BadRequest', 'fieldViolations': [{'fi eld': 'connector.subnet', 'description': 'Invalid value at \'connector.subnet\' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "private-subnet"'}]}]"> Deploying the application to App Engine... Traceback (most recent call last): File "D:\agcp\app.py", line 82, in <module> deploy\_app\_engine() ~~~~~~~~~~~~~~~~~^^ File "D:\agcp\app.py", line 71, in deploy\_app\_engine subprocess.run(['gcloud', 'app', 'deploy'], check=True) ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 556, in run with Popen(\*popenargs, \*\*kwargs) as process: ~~~~~^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1038, in \_\_init\_\_ self.\_execute\_child(args, executable, preexec\_fn, close\_fds, ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ pass\_fds, cwd, env, ^^^^^^^^^^^^^^^^^^^ ...<5 lines>... gid, gids, uid, umask, ^^^^^^^^^^^^^^^^^^^^^^ start\_new\_session, process\_group) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1550, in \_execute\_child hp, ht, pid, tid = \_winapi.CreateProcess(executable, args, ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^ # no special security ^^^^^^^^^^^^^^^^^^^^^ ...<4 lines>... cwd, ^^^^ startupinfo) ^^^^^^^^^^^^ FileNotFoundError: [WinError 2] The system cannot find the file specified - hu-devops-25 Overview Subnets Static internal IP addresses Firewalls Firewall endpoints Routes VPC network peering Private services access DNS configuration Subnets Filter Name Region Stack Type Primary IPv4 range Secondary IPv4 ranges IPv6 ranges Reserved internal ranges Gateway Private Google Access Flow logs private-subnet us-central1 IPv4 (single-stack) 10.0.1.0/24 - use this vpc details

**ChatGPT said:**

It looks like there are still two main issues you need to address:

**1. VPC Access Connector Error (Invalid subnet)**

The **VPC Access Connector** error occurs because the **subnet value** is still not being correctly provided. The **subnet** field in the **VPC Access API** needs to follow a specific format, and the error is happening because the value you provided (private-subnet) was not formatted correctly.

In your case, the correct format for the **subnet** field should be:

plaintext

CopyEdit

projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}

Since your **subnet** is in the us-central1 region, you need to pass the full resource path for the **subnet**. You can use this format:

plaintext

CopyEdit

projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}

**Correct Subnet Format:**

Replace this part of the code where you define the subnet:

python

CopyEdit

'subnet': f'{SUBNET\_NAME}', # Incorrect format

With:

python

CopyEdit

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}', # Correct format

**Updated Python Code for VPC Access Connector:**

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and region here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the vpcaccess client

def create\_vpc\_connector():

# Initialize the VPC Access API client

credentials, project = google.auth.default()

vpcaccess = build('vpcaccess', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}', # Correct subnet format

}

operation = vpcaccess.projects().locations().connectors().create(

parent=f"projects/{PROJECT\_ID}/locations/{REGION}",

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(vpcaccess, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(vpcaccess, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = vpcaccess.projects().locations().operations().get(

name=operation['name']

).execute()

if result['done']:

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Key Fix:**

* **Corrected Subnet Format**: The **subnet** field should now use the full resource path:

python

CopyEdit

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}'

**2. Fixing the gcloud Command (FileNotFoundError)**

The error FileNotFoundError: [WinError 2] The system cannot find the file specified indicates that the **gcloud** command is not available in your system’s PATH.

Follow these steps to resolve it:

**Step 1: Install Google Cloud SDK (if not installed):**

1. **Download** and **install** the Google Cloud SDK on your system.
2. Ensure that **gcloud** is added to your **PATH** during installation.

**Step 2: Verify the Installation:**

Once installed, verify if gcloud is in your PATH by running:

bash

CopyEdit

gcloud --version

If this prints the version of the gcloud command, then it's correctly installed and added to the PATH.

**Step 3: Authenticate with Google Cloud:**

If you're not already authenticated, run the following command to log in:

bash

CopyEdit

gcloud auth login

**Step 4: Add gcloud to PATH Manually (if needed):**

If gcloud still isn't recognized, you may need to manually add it to the PATH:

1. **Windows**:
   * Open **System Properties** > **Environment Variables**.
   * Find **Path** in the **System Variables** section and click **Edit**.
   * Add the directory where gcloud is installed. It should be something like:

java

CopyEdit

C:\Program Files (x86)\Google\Cloud SDK\google-cloud-sdk\bin

* + Click **OK** to save.

1. Restart your terminal or IDE.

**3. Running the Script Again**

After addressing both issues:

1. **Corrected the VPC Access Connector** configuration.
2. **Fixed the gcloud command issue**.

Run the script again:

bash

CopyEdit

python app.py

This should:

1. Create the **VPC Access Connector** correctly.
2. Deploy your app to **App Engine**.

Let me know if you face any further issues! 😊

4o

**You said:**

user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python app.py Creating VPC Access Connector 'my-vpc-connector'... An error occurred: <HttpError 400 when requesting https://vpcaccess.googleapis.com/v1/projects/manifest-zephyr-457115-v2/locations/us-central1/connectors?alt=json returned "Invalid value at 'c onnector.subnet' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "projects/manifest-zephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"". Details: "[{'@type': 't ype.googleapis.com/google.rpc.BadRequest', 'fieldViolations': [{'field': 'connector.subnet', 'description': 'Invalid value at \'connector.subnet\' (type.googleapis.com/google.cloud.vpcaccess.v 1.Connector.Subnet), "projects/manifest-zephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"'}]}]"> Deploying the application to App Engine... Traceback (most recent call last): File "D:\agcp\app.py", line 82, in <module> deploy\_app\_engine() ~~~~~~~~~~~~~~~~~^^ File "D:\agcp\app.py", line 71, in deploy\_app\_engine subprocess.run(['gcloud', 'app', 'deploy'], check=True) ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 556, in run with Popen(\*popenargs, \*\*kwargs) as process: ~~~~~^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1038, in \_\_init\_\_ self.\_execute\_child(args, executable, preexec\_fn, close\_fds, ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ pass\_fds, cwd, env, ^^^^^^^^^^^^^^^^^^^ ...<5 lines>... gid, gids, uid, umask, ^^^^^^^^^^^^^^^^^^^^^^ start\_new\_session, process\_group) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1550, in \_execute\_child hp, ht, pid, tid = \_winapi.CreateProcess(executable, args, ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^ # no special security ^^^^^^^^^^^^^^^^^^^^^ ...<4 lines>... cwd, ^^^^ startupinfo) ^^^^^^^^^^^^ FileNotFoundError: [WinError 2] The system cannot find the file specified

**ChatGPT said:**

You are still encountering two distinct errors:

**1. VPC Access Connector Error (Invalid value at 'connector.subnet')**

The **VPC Access Connector** error is caused by incorrectly specifying the subnet. The **VPC Access API** expects a full resource path for the subnet, but the format you've provided appears to be incorrect.

**Solution**: Based on the error message, the **subnet** should be provided in the correct format, and it looks like the full resource path for the **subnet** is required, including the project and region.

The **correct format** for the subnet should be:

plaintext

CopyEdit

projects/{project\_id}/regions/{region}/subnetworks/{subnet\_name}

This means that the **subnet** should be **fully qualified** with the **project**, **region**, and **subnet name**.

Here’s the **correct** code for the **subnet** field:

python

CopyEdit

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}' # Correct format

**Updated Python Code for Creating the VPC Access Connector:**

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and region here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the vpcaccess client

def create\_vpc\_connector():

# Initialize the VPC Access API client

credentials, project = google.auth.default()

vpcaccess = build('vpcaccess', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}', # Correct subnet format

}

operation = vpcaccess.projects().locations().connectors().create(

parent=f"projects/{PROJECT\_ID}/locations/{REGION}",

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(vpcaccess, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(vpcaccess, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = vpcaccess.projects().locations().operations().get(

name=operation['name']

).execute()

if result['done']:

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Key Change:**

* The **subnet** now uses the correct format, including the full resource path:

python

CopyEdit

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}' # Full resource path

**2. Fixing the gcloud Command (FileNotFoundError)**

The error FileNotFoundError: [WinError 2] The system cannot find the file specified indicates that the **gcloud** command is not available in your system's PATH.

To resolve this, follow these steps:

**Step 1: Install Google Cloud SDK (if not installed):**

1. Download and install the **Google Cloud SDK** from here.
2. During installation, ensure that **gcloud** is added to your system’s PATH (this should be automatic).

**Step 2: Verify gcloud is in the PATH**

After installation, open a new terminal and run the following command to verify that the gcloud command is available:

bash

CopyEdit

gcloud --version

If this prints the version of the gcloud command, then it's correctly installed and added to the PATH.

**Step 3: Authenticate with Google Cloud**

Make sure you're authenticated with Google Cloud:

bash

CopyEdit

gcloud auth login

**Step 4: Add gcloud to PATH Manually (if needed)**

If gcloud is still not recognized, you may need to manually add it to the PATH:

1. **Windows**:
   * Open **System Properties** > **Environment Variables**.
   * Find **Path** in the **System Variables** section and click **Edit**.
   * Add the directory where gcloud is installed. It should be something like:

java

CopyEdit

C:\Program Files (x86)\Google\Cloud SDK\google-cloud-sdk\bin

* + Click **OK** to save.

1. Restart your terminal or IDE.

**3. Run the Script Again**

After addressing both issues:

1. **Corrected the VPC Access Connector** configuration.
2. **Fixed the gcloud command issue**.

Run the script again:

bash

CopyEdit

python app.py

This should:

1. Create the **VPC Access Connector** correctly.
2. Deploy your app to **App Engine**.

Let me know if you encounter any further issues! 😊

4o

**You said:**

eNotFoundError: [WinError 2] The system cannot find the file specified user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ python app.py Creating VPC Access Connector 'my-vpc-connector'... An error occurred: <HttpError 400 when requesting https://vpcaccess.googleapis.com/v1/projects/manifest-zephyr-457115-v2/locations/us-central1/connectors?alt=json returned "Invalid value at 'c onnector.subnet' (type.googleapis.com/google.cloud.vpcaccess.v1.Connector.Subnet), "projects/manifest-zephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"". Details: "[{'@type': 't ype.googleapis.com/google.rpc.BadRequest', 'fieldViolations': [{'field': 'connector.subnet', 'description': 'Invalid value at \'connector.subnet\' (type.googleapis.com/google.cloud.vpcaccess.v 1.Connector.Subnet), "projects/manifest-zephyr-457115-v2/regions/us-central1/subnetworks/private-subnet"'}]}]"> Deploying the application to App Engine... Traceback (most recent call last): File "D:\agcp\app.py", line 82, in <module> deploy\_app\_engine() ~~~~~~~~~~~~~~~~~^^ File "D:\agcp\app.py", line 71, in deploy\_app\_engine subprocess.run(['gcloud', 'app', 'deploy'], check=True) ~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 556, in run with Popen(\*popenargs, \*\*kwargs) as process: ~~~~~^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1038, in \_\_init\_\_ self.\_execute\_child(args, executable, preexec\_fn, close\_fds, ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ pass\_fds, cwd, env, ^^^^^^^^^^^^^^^^^^^ ...<5 lines>... gid, gids, uid, umask, ^^^^^^^^^^^^^^^^^^^^^^ start\_new\_session, process\_group) ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ File "C:\Users\user121\AppData\Local\Programs\Python\Python313\Lib\subprocess.py", line 1550, in \_execute\_child hp, ht, pid, tid = \_winapi.CreateProcess(executable, args, ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^ # no special security ^^^^^^^^^^^^^^^^^^^^^ ...<4 lines>... cwd, ^^^^ startupinfo) ^^^^^^^^^^^^ FileNotFoundError: [WinError 2] The system cannot find the file specified user121@DESKTOP-JFJ21I5 MINGW64 /d/agcp $ =same error even after updating code

**ChatGPT said:**

It looks like you're still encountering two main issues:

1. **VPC Access Connector Error**: The issue with the subnet path seems persistent. The **VPC Access API** expects a **fully qualified path** for the subnet, but it appears the format is still not being accepted by the API.
2. **FileNotFoundError for gcloud**: The system is still unable to find the gcloud command, meaning it's not available in your system’s PATH.

Let's address both issues one by one.

**1. VPC Access Connector Error (Invalid value at 'connector.subnet')**

The error is happening because the **subnet** field is still not correctly formatted. The **VPC Access API** expects a **fully qualified path** to the subnet. Based on the error message you're receiving, let's try the **following approach**.

* **Format the subnet field properly**: Ensure that you're using the **full resource path** to specify the subnet, like this:

plaintext

CopyEdit

projects/{project\_id}/regions/{region}/subnetworks/{subnet\_name}

However, the exact issue you're facing is that the error is still being thrown despite providing the full path.

**Solution: Ensure you're using the right API for VPC Access and check if the subnet exists**

Make sure that:

* **VPC Access API** is enabled in your Google Cloud project.
* The **subnet** (private-subnet) exists in the correct **region** (us-central1) and the VPC network is correctly configured.

**2. Fixing the gcloud Command Not Found (FileNotFoundError)**

This error happens because the **gcloud command** is not found on your system. The gcloud command is part of the **Google Cloud SDK**, and it needs to be properly installed and added to your system’s PATH.

**Step-by-Step Solution:**

**Step 1: Install Google Cloud SDK**

1. **Download the Google Cloud SDK** from the official installation page.
2. During installation, make sure that **gcloud** is added to your **system's PATH**.

**Step 2: Verify gcloud Installation**

After installing the SDK, **open a new terminal** (important to open a new terminal window to ensure the path is refreshed).

Run:

bash

CopyEdit

gcloud --version

This command should display the version of gcloud if it’s installed correctly. If it doesn't, it means that **gcloud** is not added to your system's PATH.

**Step 3: Add gcloud to PATH Manually (If Needed)**

If gcloud still isn't recognized, follow these steps to manually add it to the PATH.

1. **On Windows**:
   * Open **System Properties** > **Environment Variables**.
   * Under **System Variables**, find **Path** and click **Edit**.
   * Add the directory where **gcloud** is installed. It should be something like:

java

CopyEdit

C:\Program Files (x86)\Google\Cloud SDK\google-cloud-sdk\bin

* + Click **OK** to save the changes.
  + Restart your terminal or IDE.

**Step 4: Authenticate with Google Cloud**

Once gcloud is available, you need to authenticate by running:

bash

CopyEdit

gcloud auth login

This will open a browser window for you to log in to your Google Cloud account.

**Updated Python Script for the VPC Access Connector Issue (Subnet Format Fix):**

After confirming that the **gcloud command** is working, try running the script again with the **fixed subnet format** for the VPC Access Connector creation.

Here's the fixed **subnet format** for the VPC Access Connector:

python

CopyEdit

import google.auth

from googleapiclient.discovery import build

from googleapiclient.errors import HttpError

import subprocess

import time

# Set your project ID, VPC, subnet, and region here

PROJECT\_ID = 'manifest-zephyr-457115-v2' # Your Google Cloud project ID

REGION = 'us-central1' # The region where your VPC and subnet reside

ZONE = 'us-central1-a' # The zone for the VM (make sure it's within 'us-central1')

INSTANCE\_NAME = 'private-ubuntu-vm' # Name of the VM (for reference)

VPC\_NAME = 'hu-devops-25' # Your VPC network name

SUBNET\_NAME = 'private-subnet' # Your private subnet name

CONNECTOR\_NAME = 'my-vpc-connector' # The name for the VPC connector

APP\_ENGINE\_ENV = 'flex' # The App Engine flexible environment

# Authenticate and create the vpcaccess client

def create\_vpc\_connector():

# Initialize the VPC Access API client

credentials, project = google.auth.default()

vpcaccess = build('vpcaccess', 'v1', credentials=credentials)

try:

# Create the VPC Access Connector

print(f"Creating VPC Access Connector '{CONNECTOR\_NAME}'...")

connector\_config = {

'name': f"projects/{PROJECT\_ID}/locations/{REGION}/connectors/{CONNECTOR\_NAME}",

'network': f'projects/{PROJECT\_ID}/global/networks/{VPC\_NAME}',

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}', # Full path for subnet

}

operation = vpcaccess.projects().locations().connectors().create(

parent=f"projects/{PROJECT\_ID}/locations/{REGION}",

body=connector\_config

).execute()

# Wait for the operation to complete

wait\_for\_operation(vpcaccess, operation)

print(f"VPC Access Connector '{CONNECTOR\_NAME}' created successfully.")

except HttpError as err:

print(f"An error occurred: {err}")

return

def wait\_for\_operation(vpcaccess, operation):

"""Wait for the operation to complete."""

print("Waiting for operation to complete...")

while True:

result = vpcaccess.projects().locations().operations().get(

name=operation['name']

).execute()

if result['done']:

if 'error' in result:

raise Exception(f"Error: {result['error']}")

else:

print("Operation completed successfully.")

break

else:

print("Operation is still in progress...")

time.sleep(10)

def deploy\_app\_engine():

"""Deploy the app to App Engine."""

try:

print("Deploying the application to App Engine...")

# Deploy the app using `gcloud app deploy`

subprocess.run(['gcloud', 'app', 'deploy'], check=True)

print("App Engine deployment completed successfully.")

except subprocess.CalledProcessError as e:

print(f"Deployment failed: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

# Step 1: Create the VPC Access Connector

create\_vpc\_connector()

# Step 2: Deploy the app to App Engine

deploy\_app\_engine()

**Summary of Fixes:**

* **Fixed subnet format**: The **subnet** field is now correctly formatted with the full resource path.

python

CopyEdit

'subnet': f'projects/{PROJECT\_ID}/regions/{REGION}/subnetworks/{SUBNET\_NAME}'

* **Addressed gcloud not found**: Ensure **Google Cloud SDK** is installed and gcloud is available in your PATH.

**Next Steps:**

1. **Ensure gcloud is installed and available** by following the installation steps.
2. **Run the script again** to create the **VPC Access Connector** and **deploy your app**.

Let me know how it goes or if you encounter any further issues! 😊